PREDICTION OF CREDIT CARD APPROVAL

Presented By

P.HEMANTH | U.PURNIMA RAMANA | I.BHAVANI | P.SAI PRASANNA

Group 4 (M.Sc. Applied Statistics)

AURORA'S DEGREE & PG COLLEGE

ABSTRACT

- To help streamline the application process and improve the approval rate by using machine learning models to predict credit card approval.

OBJECTIVE

- In this project, we will explore how various factors affect credit card approval rates and use predictive models to improve the accuracy of credit card approval predictions.

CONTENT

INTRODUCTION

- Credit cards are financial tools issued by banks that allow cardholders to borrow funds within a pre-approved limit to pay for goods and services.
- The concept of credit cards is based on the idea of a revolving line of credit, which means that users can borrow money up to a certain limit, repay it, and borrow again as needed.

Key Features of Credit Cards

- Borrowing Limit
- Interest Rates
- Building Credit History
- Security Features

How Credit Cards Work

- **Purchasing Goods and Services**:

When a cardholder makes a purchase, the bank pays the merchant on behalf of the cardholder. The cardholder then owes this amount to the bank, which they must repay by a specified due date.

- **Billing Cycle**:

Credit cards operate on a monthly billing cycle. At the end of each cycle, the cardholder receives a statement detailing their purchases, the total amount owed, the minimum payment required, and the due date.

- **Repayment**:

Cardholders can choose to pay the full balance, make the minimum payment, or pay any amount in between. Paying the full balance avoids interest charges, while paying less than the full amount leads to interest on the remaining balance.

LITERATURE REVIEW

David J. Hand and William E. Henley

Key Contribution

- Hand and Henley (1997) are well-known for their work on statistical classification methods in credit scoring.
- In their paper, they reviewed and compared traditional methods like logistic regression, discriminant analysis, and others, discussing their application in credit scoring and credit card approval processes.
- They emphasized the importance of model accuracy and the trade-offs involved in choosing different statistical techniques.

Reference: Hand, D. J., & Henley, W. E. (1997). Statistical classification methods in consumer credit scoring: A review. Journal of the Royal Statistical Society: Series A (Statistics in Society), 160(3), 523-541.

Siddiqi Naeem

Key Contribution

- Siddiqi contributed significantly to the field with his book Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring.
- He discussed the development of credit risk scorecards, which are widely used in the industry for credit card approval decisions. His work focuses on the practical aspects of implementing credit scoring models in financial institutions.

Reference: Siddiqi, N. (2012). Credit Risk Scorecards: Developing and Implementing Intelligent Credit
Scoring. John Wiley & Sons.

DATA DESCRIPTION

- Dataset of Credit Card Approval is a secondary data.
- It contains 690 rows and 16 Columns.

Numerical Variables

Age
Debt
Years employed
Credit score
Income

Categorical Variables

Gender
Married
BankCustomer
Industry
Ethnicity
PriorDefault
Employed
Driver'sLicense
Citizen
Approval status

Numerical Variables:

Age	Debt	Years Employed	Credit Score	Income
30.83	0.000	1.25	1	0
58.67	4.460	3.04	6	560
24.50	0.500	1.50	0	824
27.83	1.540	3.75	5	3
20.17	5.625	1.71	0	0

Source of the data: UC IrvineMachine Learning 'Repository

Categorical Variables :

Gender
- a-female
- b-male

Married
- u-married
- y-single
- l-divorce

Bank customer
- g- have bank account
- p-no bank account
- gg- Inactive bank account

Industry
- c- Energy
- q- Materials
- w-Industrials
- i-ConsumerDiscretionary
- aa-ConsumerStaples
- ff-Healthcare
- k-Financials
- cc-InformationTechnology
- m-CommunicationServices
- x-Utilities
- d-RealEstate
- e-Education
- j-Research
- r-Energy

Ethnicity
- v- white
- h-black
- bb-asian
- ff-latino
- j-others
- z-others
- dd-others
- n-others
- o-others

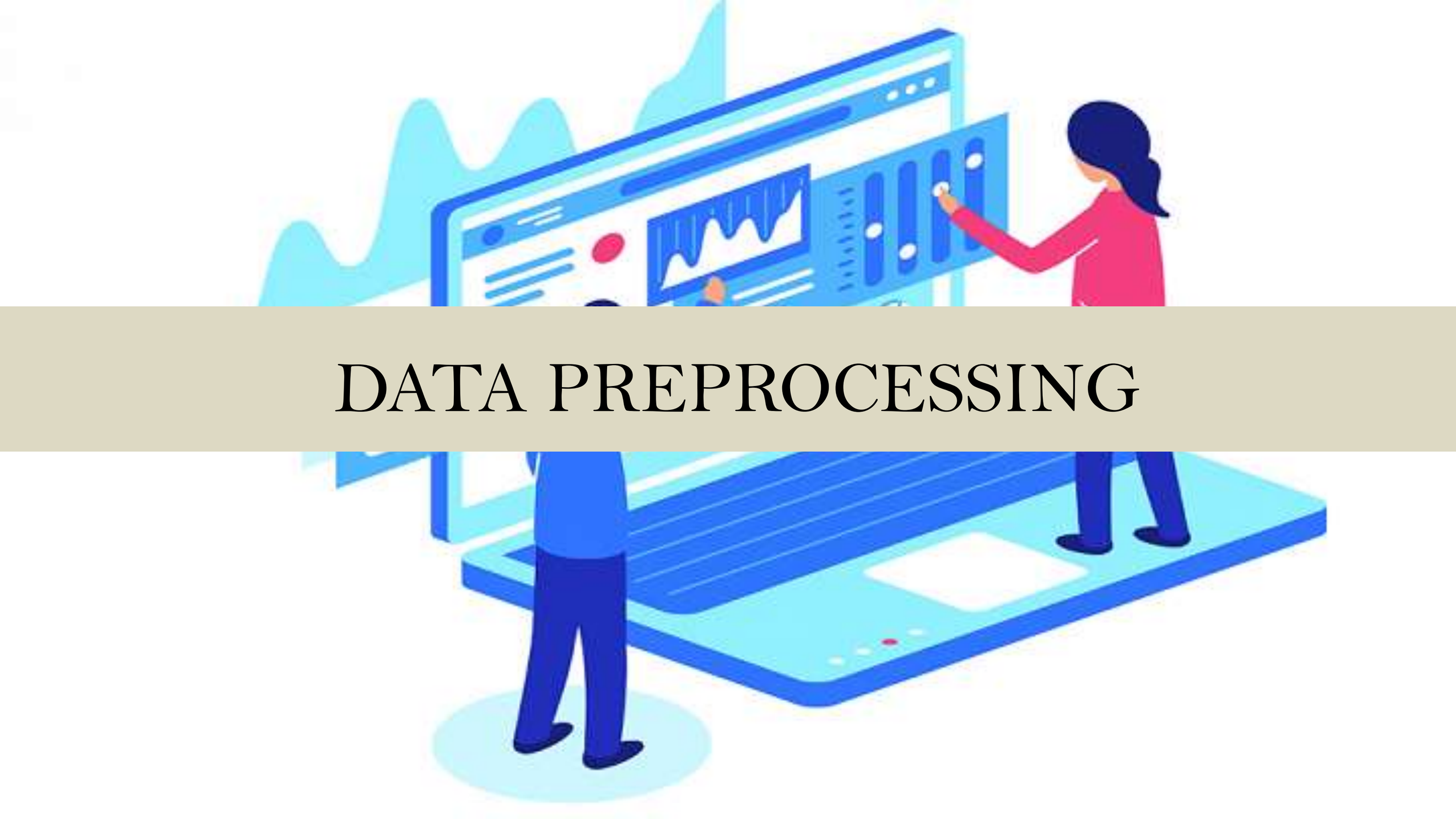
Prior default
- t-yes
- f-no

Employed
- t-yes
- f-no

Citizen
- g-birth
- s-other means
- p-Temporary

Approval status
- 1-approved
- 0- denied

Drivers Licence
- t-yes
- f-no



DATA PREPROCESSING

Data Cleaning

Gender	
Male	468
Female	210
?	12

'?' marks replaced by mode

Gender	
Male	480
Female	210

Married	
Married	519
Single	165
?	6

'?' marks replaced by mode

Married	
Married	525
Female	165

BankCustomer, Industry, Ethnicity, Citizen

? marks values were also replaced by mode

Encoding Categorical Data

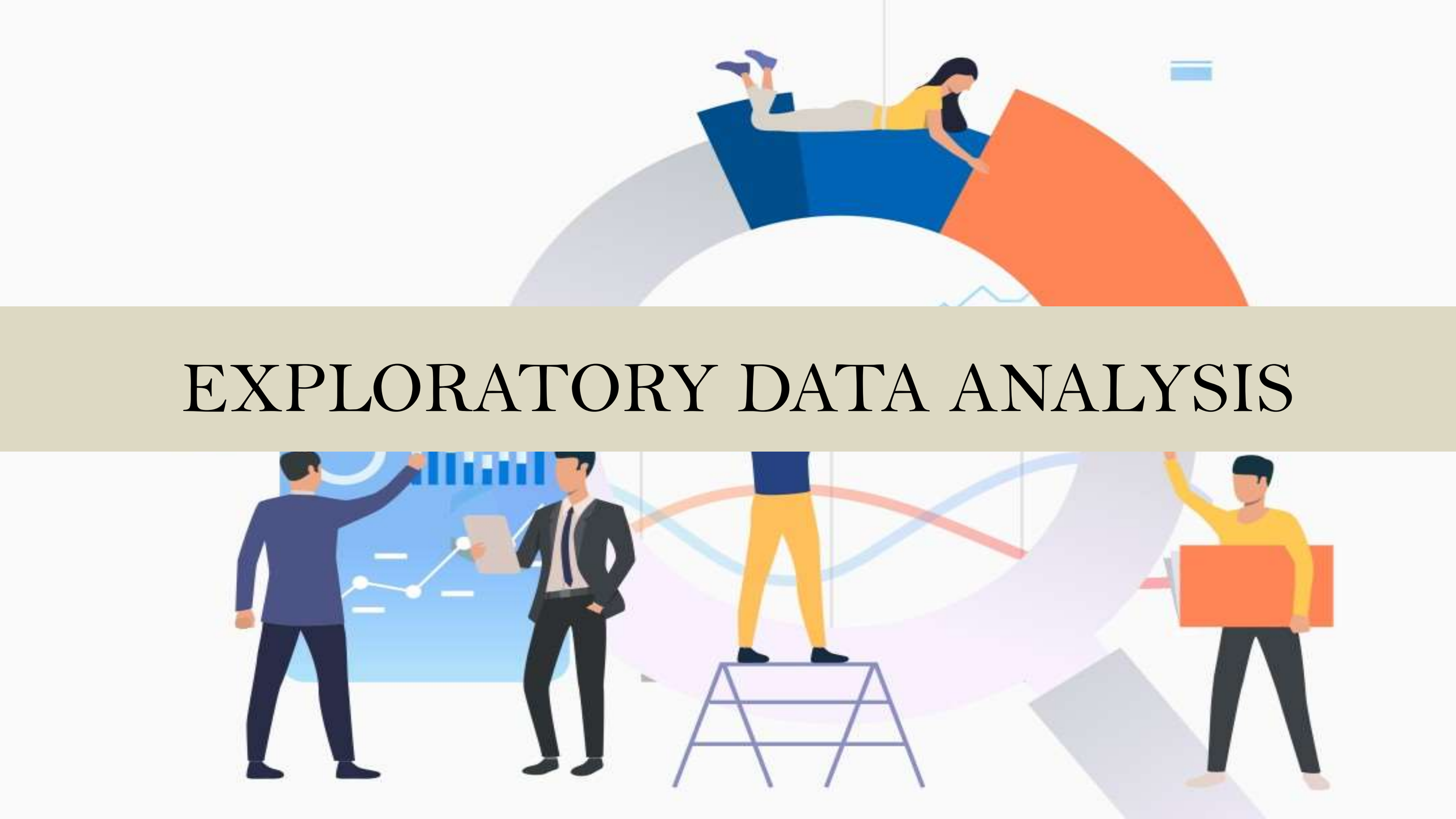
- Categorical variables often need to be converted into a numerical format so that mathematical operations can be applied.

- Dummy variables allow us to transform these categories into a form that can be easily used by the model.

Prior default	
t	1
f	0

Employee	
t	1
f	0

Driver License	
t	1
f	0

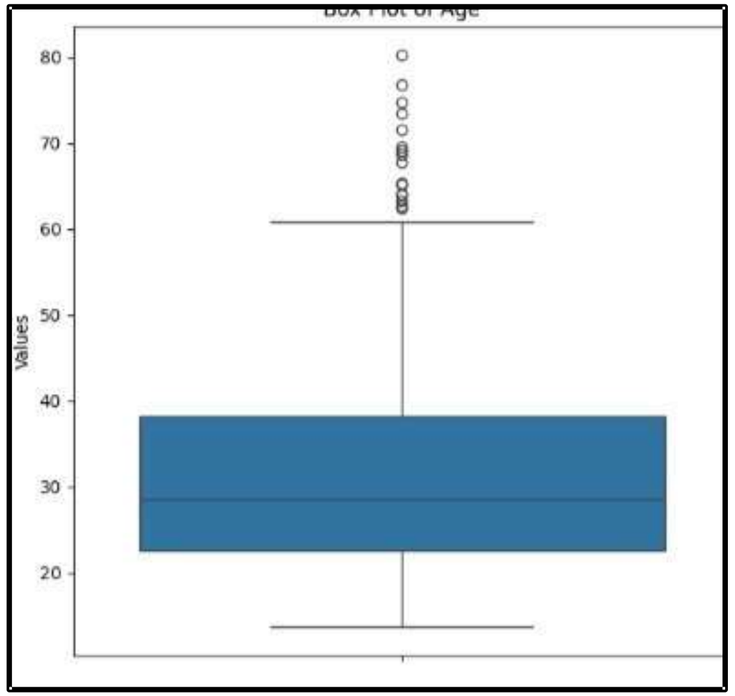


EXPLORATORY DATA ANALYSIS

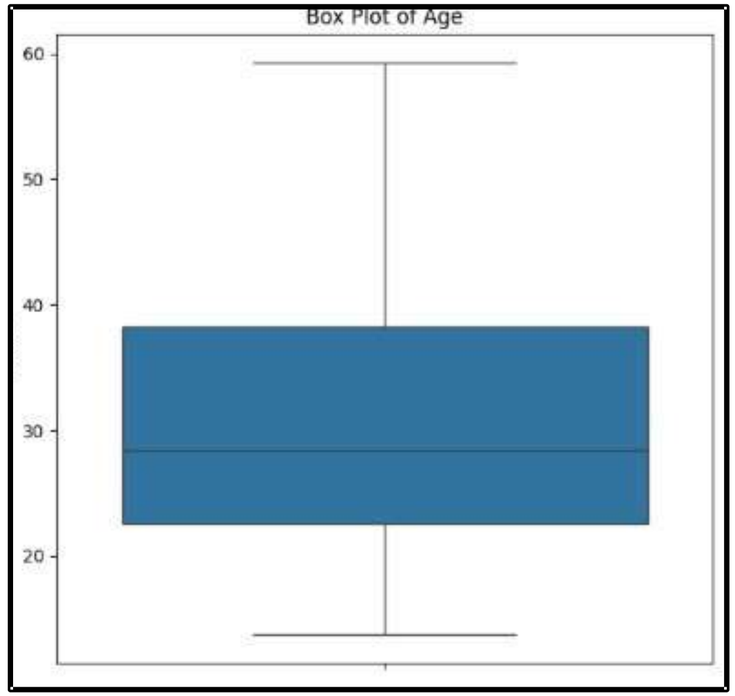
Box Plot

- In EDA part we have used Box plot for outliers detection in Numerical data .
- Approached Techniques for Handling outlier are:
- Capping to percentile
- Log transformation

Box plot for Age before Capping

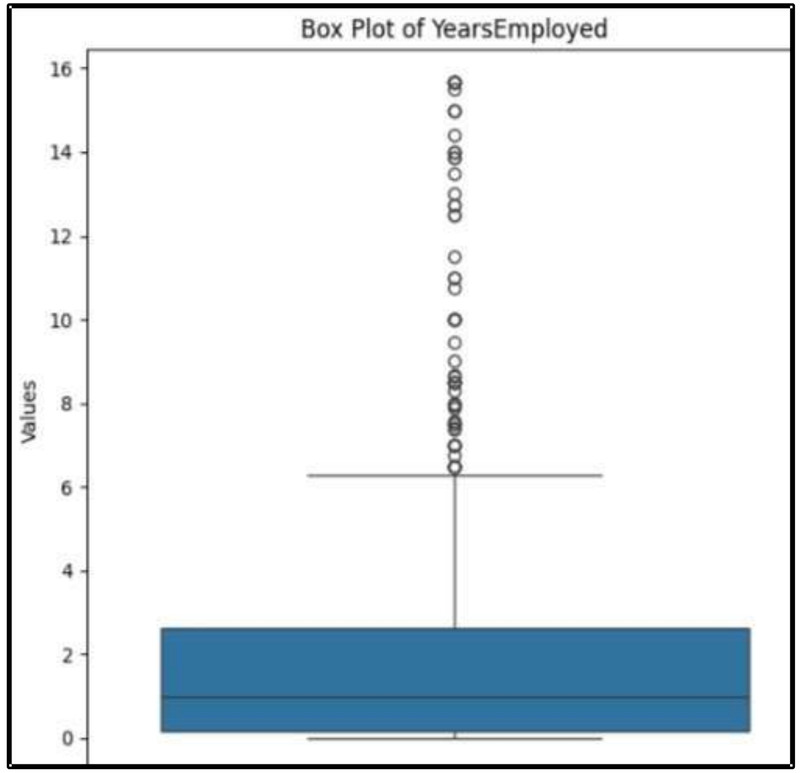


Box plot for Age after Capping

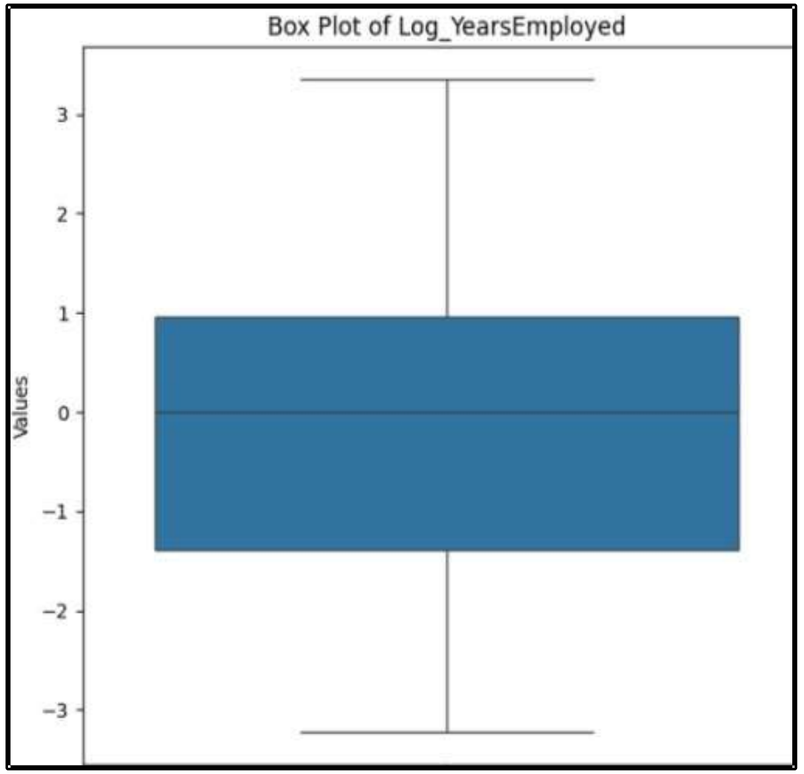


Log Transformation

Box plot for Years Employed before applying transformation



Box plot for Years Employed after applying log transformation

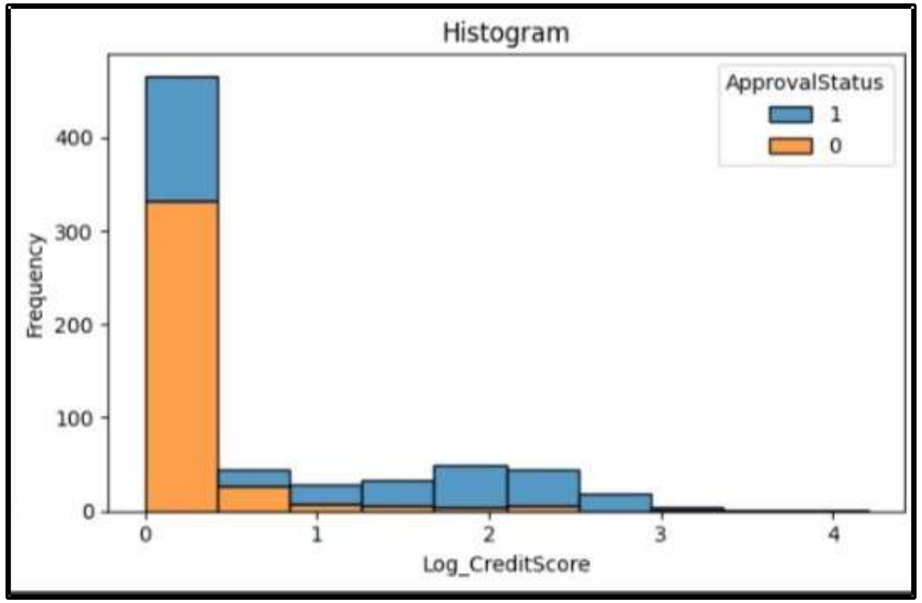


Outliers were identified and fixed

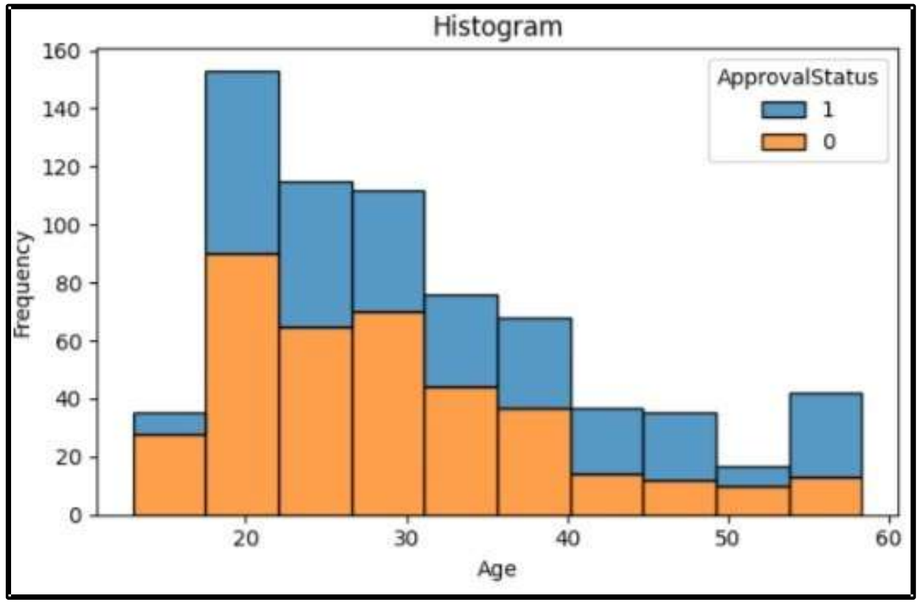
Histogram

- For log of CreditScore , approval is more than denial
- For age variable approval is more than denial

Histogram for log of credit score

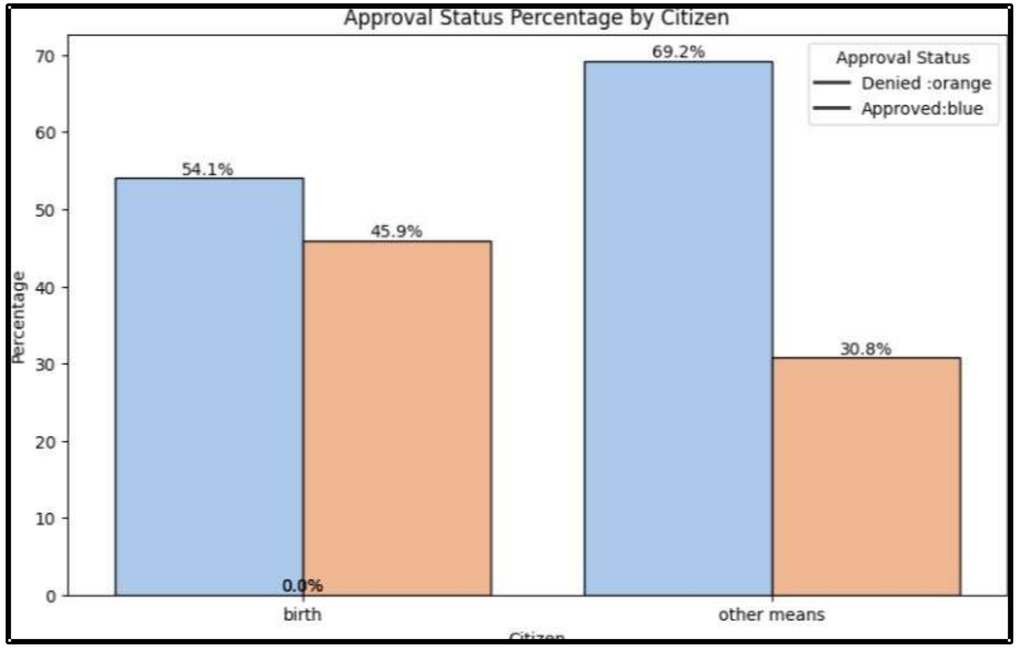


Histogram for age



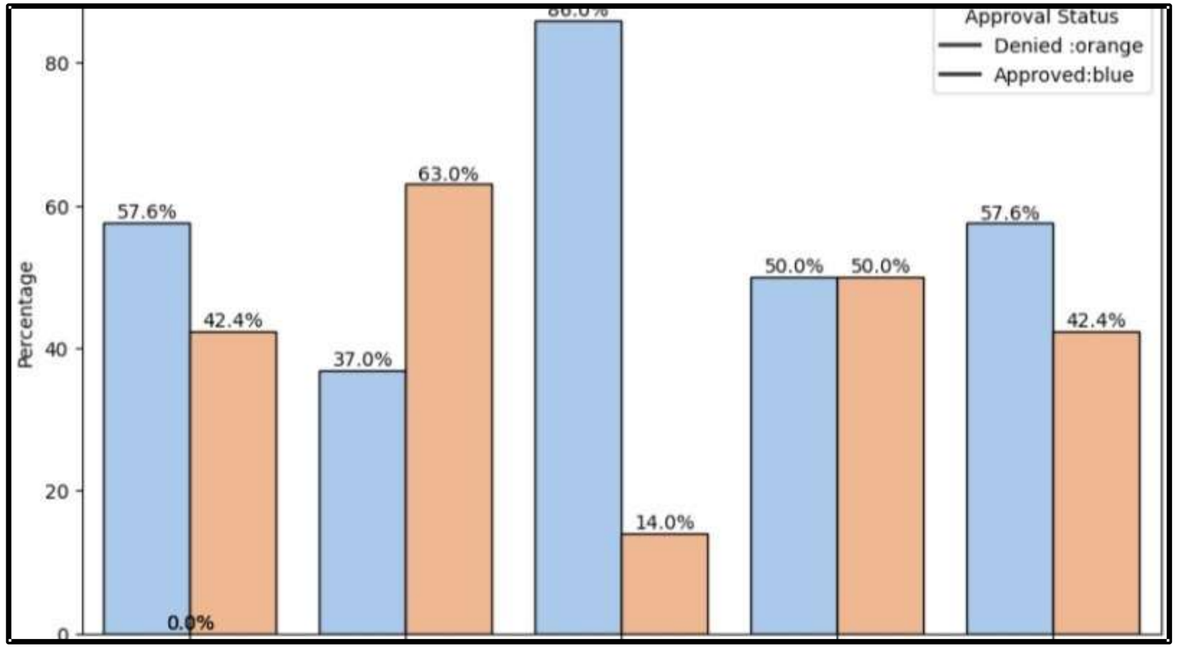
Bar Plot

Bar plot for Citizen



For the both categories by birth and by others means the approval is more than denial.

Bar plot for Ethnicity

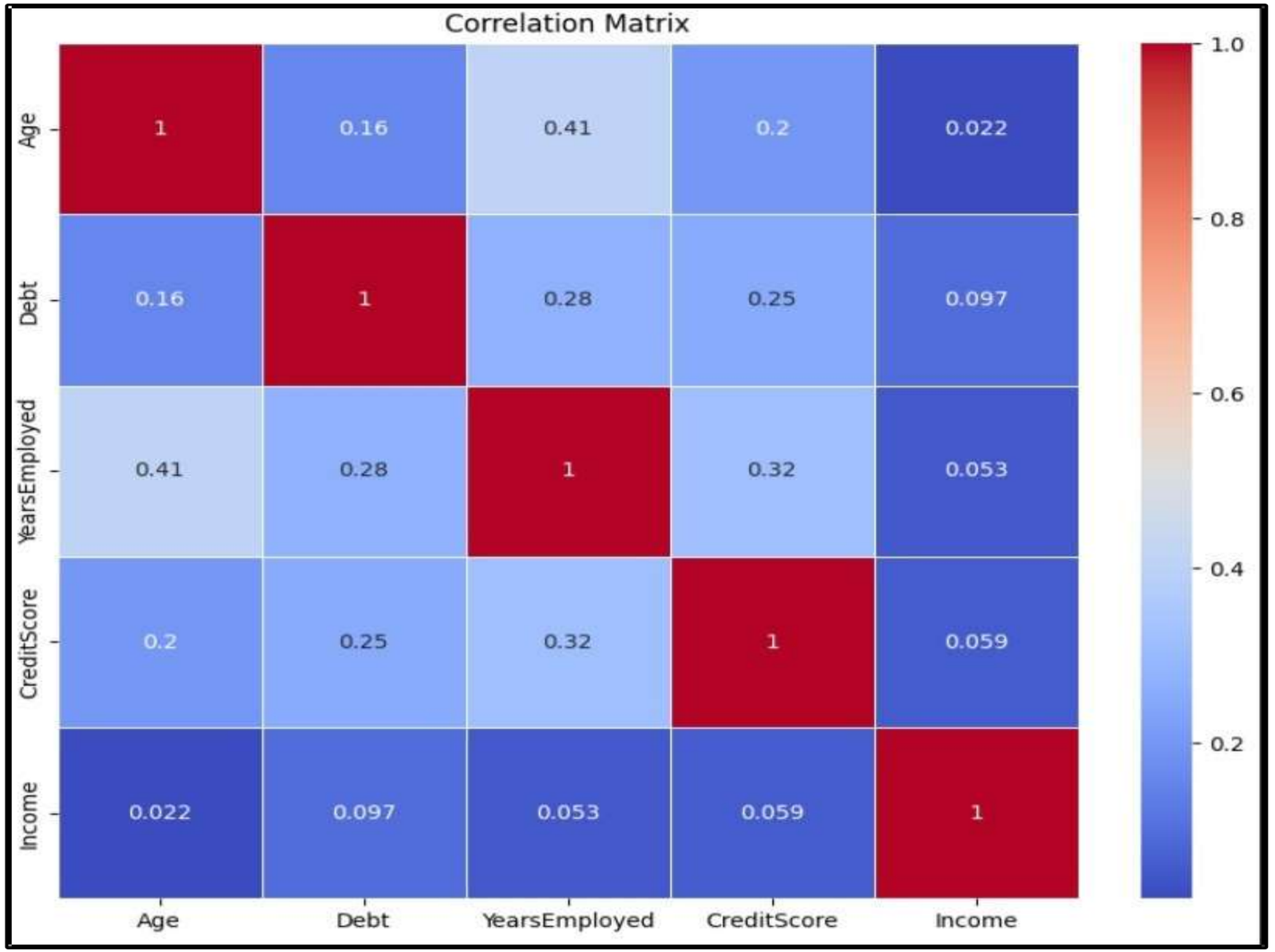


More approvals for Asian customers vs More denial for Black customers

Correlation Matrix

- Age and YearsEmployment have the highest correlation of 0.41, showing a moderate positive relationship.

- Age and Income have the lowest correlation of 0.022, indicating minimal linear relationship.



Data Reduction

For unnecessary data reduction ,we use the two features. They are

Multi Collinearity

For multi- collinearity check, we use variance inflation factor (VIF). It is a measure of the amount of multicollinearity in regression analysis. Multicollinearity exists when there is a correlation between multiple independent variables in a multiple regression model.

Feature Selection

The main goals of feature selection are to improve the model's performance, reduce overfitting, enhance generalization, and simplify the model by removing irrelevant or redundant features.

VIF values Before removing multicollinearity

0	Age	8.8
1	Debt	2.4
2	Log_Income	2.4
3	Log_YearsEmployed	1.4
4	Log_CreditScore	3.3
5	Gender_Male	3.5
6	Married_Single	inf
7	BankCustomer_NoBankAccount	inf
8	Industry_ConsumerDiscretionary	1.7
9	Industry_ConsumerStaples	2.0
10	Industry_Education	2.1
11	Industry_Energy	3.2
12	Industry_Financials	2.0
13	Industry_Healthcare	12.3
14	Industry_Industrials	2.2

15	Industry_InformationTechnology	1.8
16	Industry_Materials	2.5
17	Industry_RealEstate	1.5
18	Industry_Research	1.7
19	Industry_Transport	1.3
20	Industry_Utilities	1.8
21	Ethnicity_black	3.3
22	Ethnicity_latino	12.1
23	Ethnicity_others	2.4
24	Ethnicity_white	7.5
25	PriorDefault_1	3.4
26	Employed_1	4.0
27	DriversLicense_1	2.0
28	Citizen_other means	1.3

- Married_Single and BankCustomer_NoBankAccount have infinte VIF value
- Industry_Healthcare and Ethnicity_latinoVIF value is greater than 10

VIF values After removing multicollinearity

0	Age	8.7
1	Debt	2.4
2	Log_Income	2.3
3	Log_YearsEmployed	1.4
4	Log_CreditScore	3.3
5	Gender_Male	3.5
6	BankCustomer_NoBankAccount	1.4
7	Industry_ConsumerDiscretionary	1.7
8	Industry_ConsumerStaples	1.9
9	Industry_Education	1.9
10	Industry_Energy	3.0
11	Industry_Financials	1.8
12	Industry_Industrials	2.1
13	Industry_InformationTechnology	1.7

14	Industry_Materials	2.4
15	Industry_RealEstate	1.5
16	Industry_Research	1.4
17	Industry_Transport	1.2
18	Industry_Utilities	1.7
19	Ethnicity_black	3.3
20	Ethnicity_latino	2.0
21	Ethnicity_others	2.2
22	Ethnicity_white	7.4
23	PriorDefault_1	3.4
24	Employed_1	4.0
25	DriversLicense_1	2.0
26	Citizen_other means	1.3

After removing the Married_Single Industry_Healthcare variables all the VIF values are under 10

Feature Selection

Feature selection for significance testing involves selecting features based on their statistical significance, usually in the context of hypothesis testing.

	coef	P>|z|
const	-4.2537	0.000
Age	0.0126	0.440
Debt	0.0230	0.544
Log_Income	0.1218	0.027
Log_YearsEmployed	-0.0466	0.714
Log_CreditScore	0.3605	0.199
Gender_Male	0.1521	0.673
BankCustomer_NoBankAccount	-1.1096	0.005
Industry_ConsumerDiscretionary	0.3048	0.734
Industry_ConsumerStaples	-0.1183	0.885
Industry_Education	0.2847	0.794
Industry_Energy	0.5535	0.450
Industry_Financials	0.2076	0.806
Industry_Industrials	0.9741	0.220
Industry_InformationTechnology	1.2662	0.194
Industry_Materials	0.2366	0.765
Industry_RealEstate	-0.1510	0.874
Industry_Research	1.2006	0.667
Industry_Transport	1.3280	0.303
Industry_Utilities	2.3922	0.033
Ethnicity_black	0.2953	0.653
Ethnicity_latino	-1.0498	0.339
Ethnicity_others	1.3496	0.230
Ethnicity_white	0.3359	0.588
PriorDefault_1	3.8222	0.000
Employed_1	0.3105	0.497
DriversLicense_1	-0.0705	0.827
Citizen_other means	0.7559	0.200

"p ≤ 0.05: statistically significant;
p > 0.05: not statistically significant."

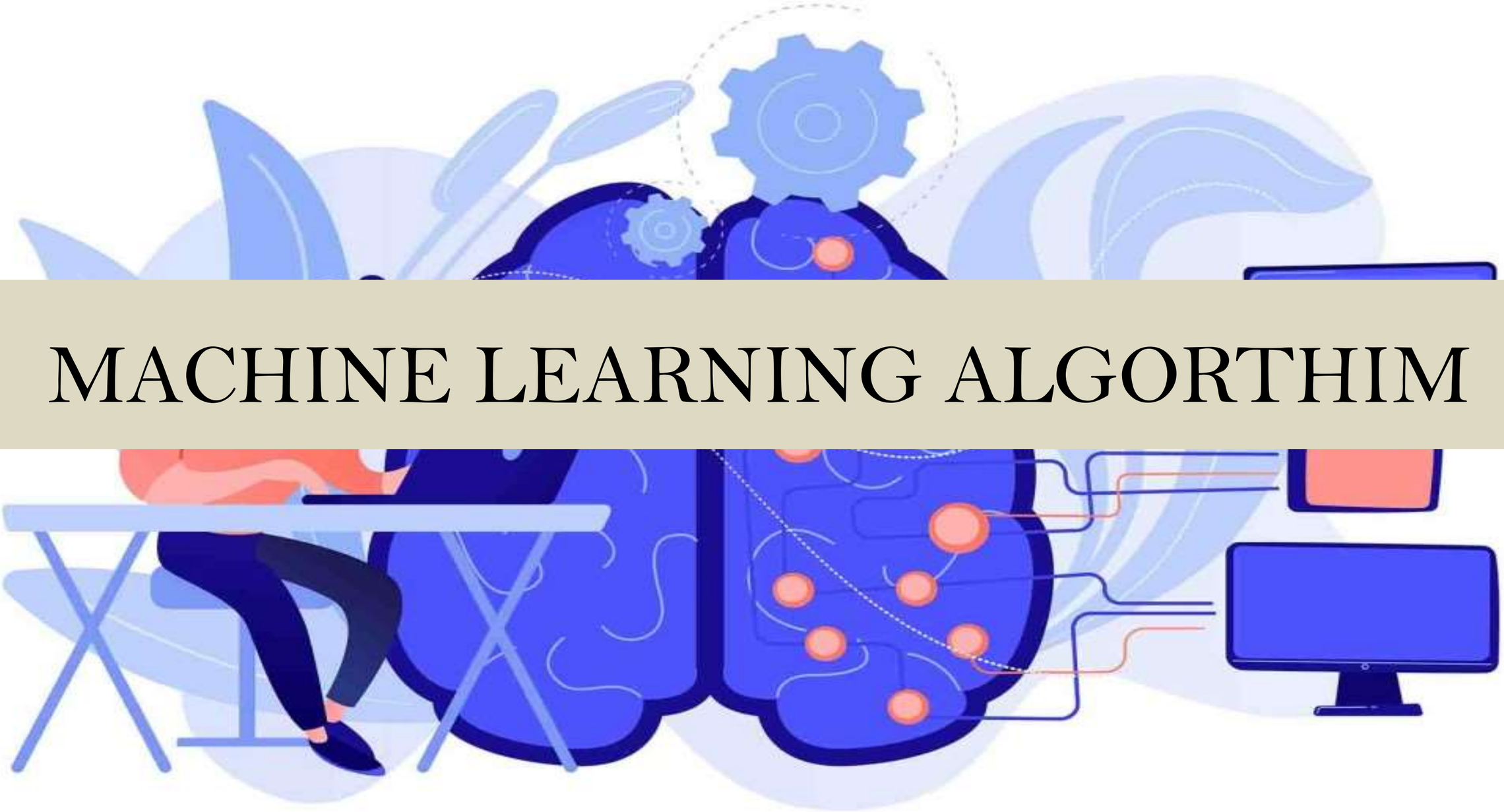
Feature Selection

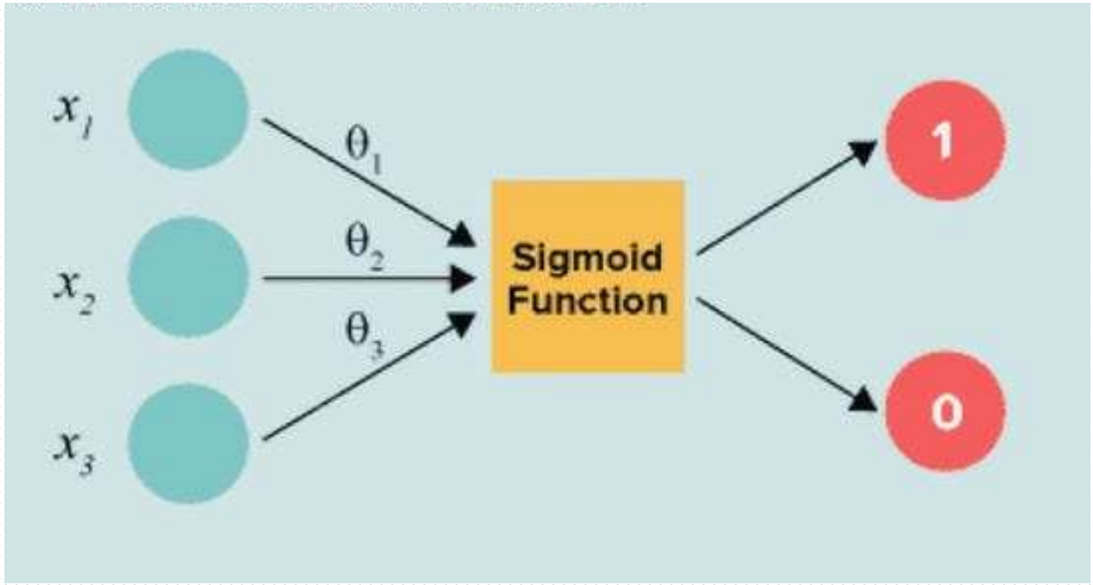
- The goal is to identify which features have a statistically significant relationship with the target variable, indicating that they might be important predictors in a model.

- Below are the Significant features identified through feature selection.

```
Log_Income
Log_CreditScore
Industry_Industrials
Industry_InformationTechnology
Industry_Transport
Industry_Utilities
Ethnicity_others
PriorDefault_1
```

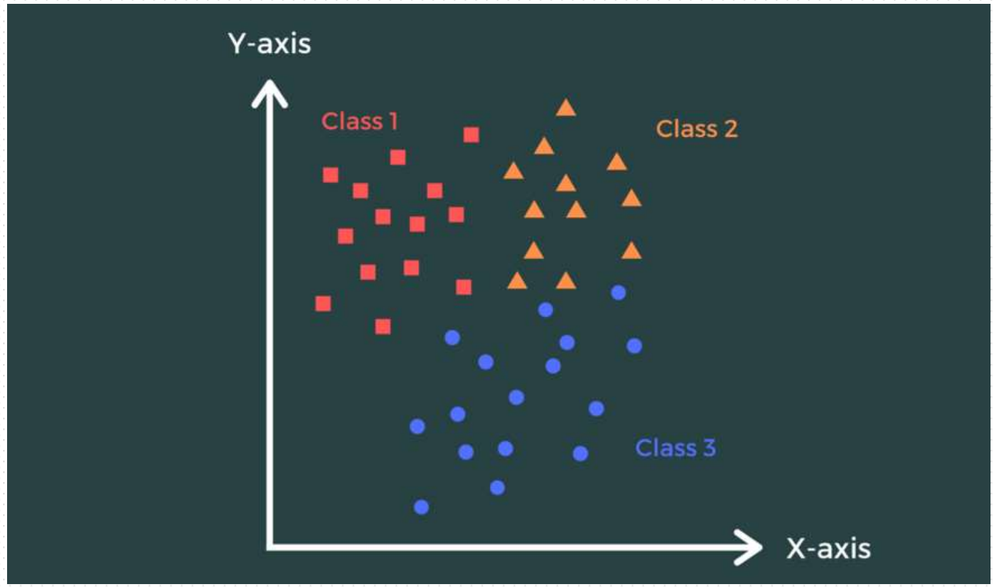
MACHINE LEARNING ALGORTHIM





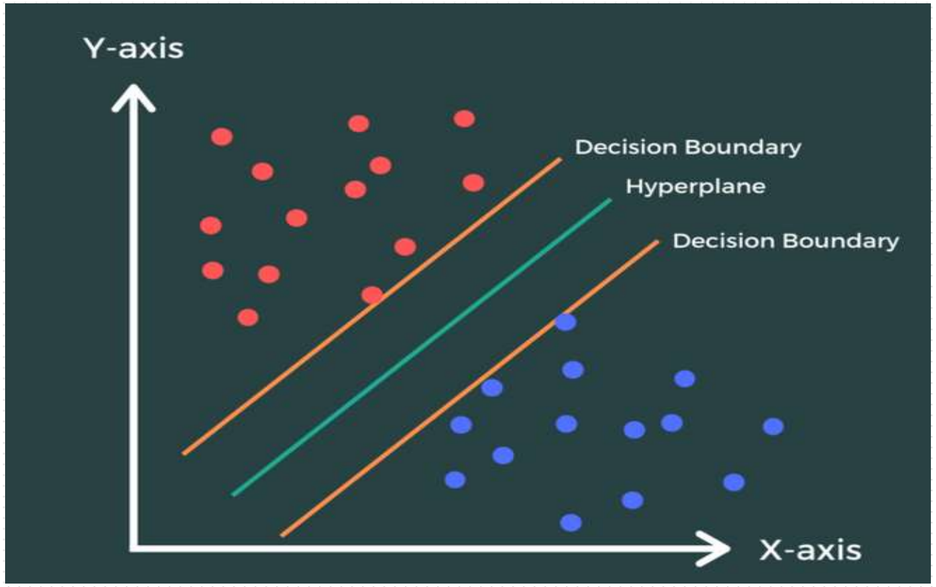
Logistic Regression

A statistical model used for binary classification that predicts the probability of a categorical outcome based on one or more predictor variables.



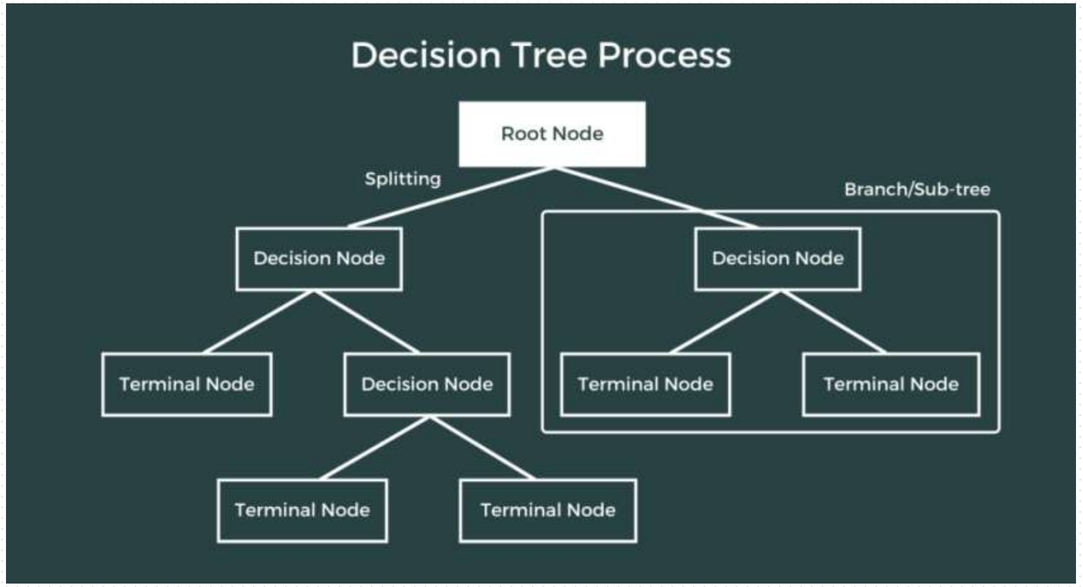
KNN

A classification algorithm that assigns labels based on the majority vote of nearest neighbors in the feature space.



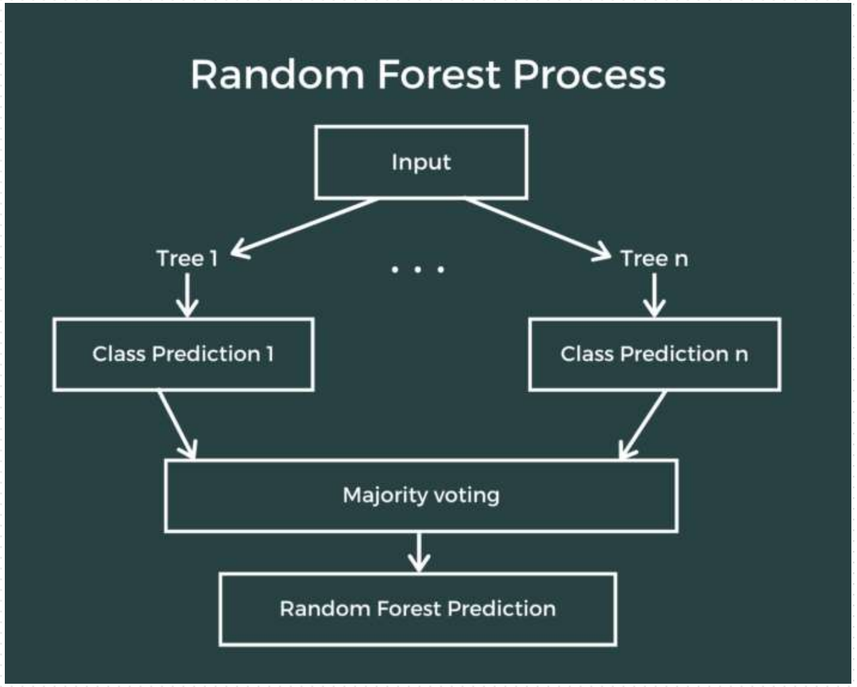
SVM

A classification algorithm that finds the optimal hyperplane to separate data into distinct classes.



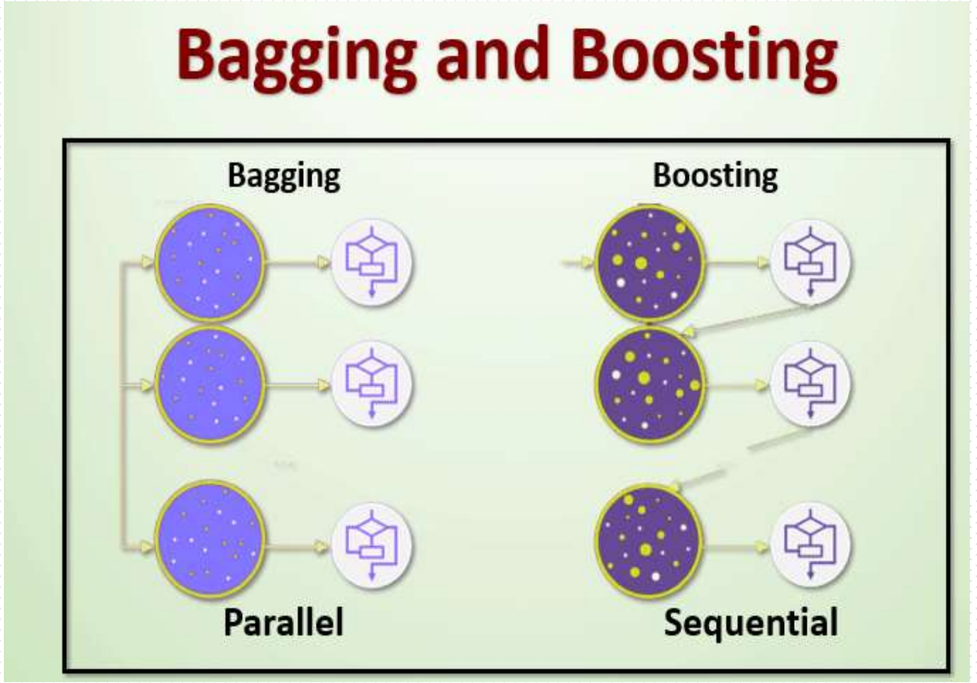
Decision Tree

A model that splits data into subsets based on feature values to make predictions or decisions.



Random Forest

An ensemble learning method that builds multiple decision trees and aggregates their predictions for improved accuracy and robustness..



Bagging

Aggregates predictions from multiple models to improve accuracy and reduce variance

Boosting

Sequentially builds models to correct errors of previous ones, enhancing predictive performance.

80-20 Train-Test Split

Machine Learning Models	All Data	Data with No Multicollinearity Features	Data with Significant Features
Logistic Regression	0.841	0.841	0.848
K-Nearest Neighbour	0.783	0.783	0.812
Support Vector Machine	0.826	0.826	0.841
Decision Tree	0.768	0.768	0.804
Random Forest	0.841	0.841	0.819
Bagging	0.797	0.797	0.826
Boosting	0.761	0.761	0.797

75-25 Train-Test Split

Machine Learning Models	All Data	Data with No Mullticolinearity Features	Data with Significance Features
Logistic Regression	0.844	0.844	0.861
K-Nearest Neighbour	0.798	0.798	0.821
Support Vector Machine	0.832	0.832	0.855
Decision Tree	0.775	0.775	0.827
Random Forest	0.844	0.844	0.844
Bagging	0.855	0.855	0.838
Boosting	0.786	0.786	0.809

70-30 Train-Test Split

Machine Learning Models	All Data	Data with No Multicollinearity Features	Data with Significant Features
Logistic Regression	0.841	0.841	0.855
K-Nearest Neighbour	0.807	0.807	0.831
Support Vector Machine	0.831	0.831	0.855
Decision Tree	0.812	0.812	0.841
Random Forest	0.86	0.86	0.841
Bagging	0.865	0.865	0.845
Boosting	0.812	0.812	0.836

60-40 Train-Test Split

Machine Learning Models	All Data	Data with No Mullticolinearity Features	Data with Significant Features
Logistic Regression	0.826	0.826	0.859
K-Nearest Neighbour	0.808	0.808	0.859
Support Vector Machine	0.855	0.855	0.859
Decision Tree	0.801	0.801	0.859
Random Forest	0.862	0.862	0.859
Bagging	0.822	0.822	0.859
Boosting	0.801	0.801	0.859

Algorithms Comparison

Data	Over All Data				Data With No Multicolinearity Features				Data With Significant Features			
ML Algorithms	80-20	75-25	70-30	60-40	80-20	75-25	70-30	60-40	80-20	75-25	70-30	60-40
Logistic regression	0.841	0.844	0.841	0.826	0.841	0.844	0.841	0.826	0.848	0.861	0.855	0.859
K-Nearest Neighbour	0.783	0.798	0.807	0.808	0.783	0.798	0.807	0.808	0.812	0.821	0.831	0.859
Support vector machine	0.826	0.832	0.831	0.855	0.826	0.832	0.831	0.855	0.841	0.855	0.855	0.859
Decision Tree	0.768	0.775	0.812	0.801	0.768	0.775	0.812	0.801	0.804	0.827	0.841	0.859
Random Forest	0.841	0.844	0.86	0.862	0.841	0.844	0.86	0.862	0.819	0.844	0.841	0.859
Bagging	0.797	0.855	0.865	0.822	0.797	0.855	0.865	0.822	0.826	0.838	0.845	0.859
Boosting	0.761	0.786	0.812	0.801	0.761	0.786	0.812	0.801	0.797	0.809	0.836	0.859

•**Overall Performance**: The models generally show better performance with the "Data with Significant Features" and "Data with No Multicollinearity Features" compared to "All Data."

CONCLUSION

Conclusion

- Overall, for our credit card approval project, the Random Forest and Bagging models performed the best in most scenarios, making them strong candidates for deployment.

- Logistic Regression also offers reliable results across all datasets and ratios. Selecting the appropriate model will depend on the specific data characteristics and performance requirements of the project.

- The 60-40 split is results as the ‘Best Spilt’

- The dataset with significant features maintains the highest accuracy compared to the other two datasets. Therefore, industries such as Transport, Utilities, and Information Technology have a higher likelihood of credit card approval.".

THANK YOU

P.Hemanth
U.Purnima Ramana
I.Bhavani
P.Sai Prasanna

APPENDIX

LOADING LIBRARIES

```
[ ]  #Imports
     import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
```

```
[ ] data.head()
```

	Gender	Age	Debt	Married	BankCustomer	Industry	Ethnicity	YearsEmployed	PriorDefault	Employed	CreditScore	DriversLicense	Citizen	ZipCode	Income	ApprovalStatus
0	b	30.83	0.000	u	g	w	v	1.25	t	t	1	f	g	00202	0	+
1	a	58.67	4.460	u	g	q	h	3.04	t	t	6	f	g	00043	560	+
2	a	24.50	0.500	u	g	q	h	1.50	t	f	0	f	g	00280	824	+
3	b	27.83	1.540	u	g	w	v	3.75	t	t	5	t	g	00100	3	+
4	b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	00120	0	+

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Gender          690 non-null    object 
 1   Age             690 non-null    object 
 2   Debt            690 non-null    float64
 3   Married         690 non-null    object 
 4   BankCustomer    690 non-null    object 
 5   Industry        690 non-null    object 
 6   Ethnicity       690 non-null    object 
 7   YearsEmployed   690 non-null    float64
 8   PriorDefault    690 non-null    object 
 9   Employed        690 non-null    object 
 10  CreditScore     690 non-null    int64  
 11  DriversLicense  690 non-null    object 
 12  Citizen         690 non-null    object 
 13  ZipCode         690 non-null    object 
 14  Income          690 non-null    int64  
```

```
#To get the data description
data.describe()
```

	Debt	YearsEmployed	CreditScore	Income
count	690.000000	690.000000	690.00000	690.000000
mean	4.758725	2.223406	2.40000	1017.385507
std	4.978163	3.346513	4.86294	5210.102598
min	0.000000	0.000000	0.00000	0.000000
25%	1.000000	0.165000	0.00000	0.000000
50%	2.750000	1.000000	0.00000	5.000000
75%	7.207500	2.625000	3.00000	395.500000
max	28.000000	28.500000	67.00000	100000.000000

```
#To check missing values in each column
for i in range(data.shape[1]):
  print(data.iloc[:,i].unique())
  print(data.iloc[:,i].value_counts())
```

```
['b' 'a' '?']
Gender
b    468
a    210
?     12
```

```
#Gender
#To convert 'a','b',with 'Female','Male'
#To replace'?', with mode of the column,'Male'
data['Gender'].replace({
    'a': 'Female',
    'b': 'Male',
    '?': 'Male'
}, inplace=True)
```

```
(data.Gender.value_counts())
```

Gender	count
Male	480
Female	210

dtype: int64

```
#Married
#To convert 'u', with Married,'y',with Single
#replacing 'l'(divorce),with'Single ,since 'l' is having only two values
#To replace missing value '?',with mode of the column,'Married'
data['Married'].replace({
    'u': 'Married',
    'y': 'Single',
    'l': 'Single',
    '?': 'Married'
}, inplace=True)
```

```
(data.Married.value_counts())
```

Married	count
Married	525
Single	165

dtype: int64

```
[ ] #BankCustomer
    #To convert 'g' to HavingBankAccount,'p' to NoBankAccount and 'gg (closed account)' to NoBankAccount
    # to replace '?' with mode of the column 'HavingBankAccount
    data['BankCustomer'].replace({
        'g': 'HavingBankAccount',
        'gg': 'NoBankAccount',
        '?': 'HavingBankAccount',
        'p': 'NoBankAccount'
    }, inplace=True)
```

```
(data.BankCustomer.value_counts())
```

BankCustomer	count
HavingBankAccount	525
NoBankAccount	165

dtype: int64

```
[ ] data['ApprovalStatus'].replace({'+':'1','-':'0'},inplace=True)
```

```
] (data.ApprovalStatus.value_counts())
```

ApprovalStatus	count
0	383
1	307

dtype: int64

```
[ ] #Employed
    #replace't'(Employed)to '1','f'(UnEmployed)to '0'
    data['Employed'].replace({'t': 1, 'f': 0}, inplace=True)
```

```
(data.Employed.value_counts())
```

Employed	count
0	395
1	295

dtype: int64

```
[ ] #PriorDefault
    #replace't'(true)to '1','f'(false)to '0'
    data['PriorDefault'].replace({'t':'1','f':'0'},inplace=True)
```

```
[ ] (data.PriorDefault.value_counts())
```

PriorDefault	count
1	361
0	329

dtype: int64

```
#industry
#To replace['c'='Energy,'q'='Materials','w'='Industrials','i'='ConsumerDiscretionary','aa'='ConsumerStaples','ff'='Healthcare','k'='Financials',
#'cc'='InformationTechnology','m'='CommunicationServices','x'='Utilities','d'='RealEstate','e'='Education','j'='Research','r'='Transport','?'='Energy
replacement_dict = {
    'c': 'Energy',
    'q': 'Materials',
    'w': 'Industrials',
    'i': 'ConsumerDiscretionary',
    'aa': 'ConsumerStaples',
    'ff': 'Healthcare',
    'k': 'Financials',
    'cc': 'InformationTechnology',
    'm': 'CommunicationServices',
    'x': 'Utilities',
    'd': 'RealEstate',
    'e': 'Education',
    'j': 'Research',
    '?': 'Transport',
    'r': 'Energy'
}
data['Industry'].replace(replacement_dict, inplace=True)
```

Industry	count
Energy	140
Materials	78
Industrials	64
sumerDiscretionary	59
onsumerStaples	54
Healthcare	53
Financials	51
rmationTechnology	41
munication Services	38
Utilities	38
RealEstate	30
Education	25
Research	10
Transport	9

: int64

```
#Ethnicity
#replace 'v'with'White,'h'with'black','bb','asian','ff'with 'latino',and 'j','z','dd','n','o'with 'others'
#replace ? with mode ,'White'
data['Ethnicity'].replace({
    'v': 'white',
    'h': 'black',
    'bb': 'asian',
    'ff': 'latino',
    'j': 'others',
    'z': 'others',
    'dd': 'others',
    'n': 'others',
    'o': 'others',
    '?': 'white'
}, inplace=True)
```

```
] (data.Ethnicity.value_counts())
```

Ethnicity	count
white	408
black	138
asian	59
latino	57
others	28

dtype: int64

```
[ ] #Citizen
    #to replace 'g' to 'birth','s'to 'other means','p'to 'Temporary'
    #to replace 'Temporary' to 'other means' ,since Temporary is having only 8 values
    #since Temporary is very small we can  reperesent with 'other means'
    data['Citizen'].replace({'g':'birth','s':'other means','p':'Temporary'},inplace=True)
    data['Citizen'].replace({'Temporary':'other means'},inplace=True)
```

```
(data.Citizen.value_counts())
```

Citizen	count
birth	625
other means	65

dtype: int64

```
[ ] #Employed
    #replace't'(Employed)to '1','f'(UnEmployed)to '0'
    data['Employed'].replace({'t': 1, 'f': 0}, inplace=True)
```

```
[ ] (data.Employed.value_counts())
```

Employed	count
0	395
1	295

dtype: int64

Logistic Regression

```
[ ] Columns_to_dummify = ['Gender','Married','BankCustomer','Industry','Ethnicity','PriorDefault','Employed','DriversLicense','Citizen','ApprovalStatus']
```

```
[ ] data_dummies = pd.get_dummies(data, columns=Columns_to_dummify, drop_first=True,dtype=int)
    data_dummies_clean=data_dummies.drop(['ZipCode','YearsEmployed','CreditScore','Income' ],axis=1)
```

```
data_dummies_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 30 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Age                             690 non-null    float64
 1   Debt                            690 non-null    float64
 2   Log_Income                      690 non-null    float64
 3   Log_YearsEmployed               690 non-null    float64
 4   Log_CreditScore                 690 non-null    float64
 5   Gender_Male                     690 non-null    int64
 6   Married_Single                  690 non-null    int64
 7   BankCustomer_NoBankAccount      690 non-null    int64
 8   Industry_ConsumerDiscretionary  690 non-null    int64
 9   Industry_ConsumerStaples        690 non-null    int64
 10  Industry_Education              690 non-null    int64
 11  Industry_Energy                 690 non-null    int64
 12  Industry_Financials             690 non-null    int64
 13  Industry_Healthcare             690 non-null    int64
 14  Industry_Industrials            690 non-null    int64
 15  Industry_InformationTechnology  690 non-null    int64
 16  Industry_Materials              690 non-null    int64
 17  Industry_RealEstate             690 non-null    int64
 18  Industry_Research               690 non-null    int64
 19  Industry_Transport              690 non-null    int64
 20  Industry_Utilities              690 non-null    int64
 21  Ethnicity_black                 690 non-null    int64
 22  Ethnicity_latino                690 non-null    int64
 23  Ethnicity_others                690 non-null    int64
 24  Ethnicity_white                 690 non-null    int64
 25  PriorDefault_1                  690 non-null    int64
 26  Employed_1                      690 non-null    int64
 27  DriversLicense_1                690 non-null    int64
 28  Citizen_other means             690 non-null    int64
 29  ApprovalStatus_1                690 non-null    int64
dtypes: float64(5), int64(25)
memory usage: 161.8 KB
```

```
[ ] data_dummies_clean.to_pickle('cc_cleandata.pkl')
```

```
[ ] data_dummies_clean=pd.read_pickle('cc_cleandata.pkl')
```

```
[ ] X_clean=data_dummies_clean.drop(['ApprovalStatus_1'],axis=1)
    Y=data_dummies_clean['ApprovalStatus_1']
    from sklearn.model_selection import train_test_split
    X_clean_train, X_clean_test, Y_train, Y_test=train_test_split(X_clean,Y, test_size=0.30,train_size=0.70)
```

```
# Import library for VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calc_vif(X):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i).round(1) for i in range(X.shape[1])]

    return(vif)

calc_vif(data_dummies_clean.drop(['ApprovalStatus_1'],axis=1))
```

```python
# Import library for VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calc_vif(X):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i).round(1) for i in range(X.shape[1])]

    return(vif)

calc_vif(data_dummies_clean.drop(['ApprovalStatus_1'],axis=1))
```

1	Debt	2.4
2	Log_Income	2.4
3	Log_YearsEmployed	1.4
4	Log_CreditScore	3.3
5	Gender_Male	3.5
6	Married_Single	inf
7	BankCustomer_NoBankAccount	inf
8	Industry_ConsumerDiscretionary	1.7
9	Industry_ConsumerStaples	2.0
10	Industry_Education	2.1
11	Industry_Energy	3.2
12	Industry_Financials	2.0
13	Industry_Healthcare	12.3
14	Industry_Industrials	2.2
15	Industry_InformationTechnology	1.8
16	Industry_Materials	2.5
17	Industry_RealEstate	1.5
18	Industry_Research	1.7
19	Industry_Transport	1.3
22	Ethnicity_latino	12.1
23	Ethnicity_others	2.4
24	Ethnicity_white	7.5
25	PriorDefault_1	3.4
26	Employed_1	4.0
27	DriversLicense_1	2.0
28	Citizen_other means	1.3

```
calc_vif(data_dummies_clean.drop(['Married_Single','Industry_Healthcare','ApprovalStatus_1' ],axis=1
```

	variables	VIF
0	Age	8.7
1	Debt	2.4
2	Log_Income	2.3
3	Log_YearsEmployed	1.4
4	Log_CreditScore	3.3
5	Gender_Male	3.5
6	BankCustomer_NoBankAccount	1.4
7	Industry_ConsumerDiscretionary	1.7
8	Industry_ConsumerStaples	1.9
9	Industry_Education	1.9
10	Industry_Energy	3.0
11	Industry_Financials	2.0
12	Industry_Healthcare	12.3
13	Industry_Industrials	2.2
14	Industry_InformationTechnology	1.8
15	Industry_Materials	2.5
16	Industry_RealEstate	1.5
17	Industry_Research	1.7
18	Industry_Transport	1.3
19	Industry_Utilities	1.8
20	Ethnicity_black	3.3
21	Ethnicity_latino	12.1
22	Ethnicity_others	2.4
23	Ethnicity_white	7.5
24	PriorDefault_1	3.4
25	Employed_1	4.0
26	DriversLicense_1	2.0
27	Citizen_other means	1.3

```
calc_vif(data_dummies_clean.drop(['Married_Single','Industry_Healthcare','ApprovalStatus_1' ],axis=1))
```

	variables	VIF
0	Age	8.7
1	Debt	2.4
2	Log_Income	2.3
3	Log_YearsEmployed	1.4
4	Log_CreditScore	3.3
5	Gender_Male	3.5
6	BankCustomer_NoBankAccount	1.4
7	Industry_ConsumerDiscretionary	1.7
8	Industry_ConsumerStaples	1.9
9	Industry_Education	1.9
10	Industry_Energy	3.0
11	Industry_Financials	1.8
12	Industry_Industrials	2.1
13	Industry_InformationTechnology	1.7
14	Industry_Materials	2.4
15	Industry_RealEstate	1.5
16	Industry_Research	1.4
17	Industry_Transport	1.2
18	Industry_Utilities	1.7
19	Ethnicity_black	3.3
20	Ethnicity_latino	2.0
21	Ethnicity_others	2.2
22	Ethnicity_white	7.4
23	PriorDefault_1	3.4
24	Employed_1	4.0
25	DriversLicense_1	2.0
26	Citizen_other means	1.3

```
[ ] data_dummies_nomulti=data_dummies_clean.drop(['Married_Single','Industry_Healthcare' ],axis=1)
```

```
[ ] X_nomulti=data_dummies_nomulti.drop(['ApprovalStatus_1' ],axis=1)
    from sklearn.model_selection import train_test_split
    X_nomulti_train, X_nomulti_test, Y_train, Y_test=train_test_split(X_nomulti,Y, test_size=0.30,train_size=0.70)
```

```
import statsmodels.api as sm

X_nomulti_train_cns=sm.add_constant(X_nomulti_train)
model = sm.Logit(Y_train,X_nomulti_train_cns ).fit()

# Print the summary to get p-values
print(model.summary())
```

Logit Regression Results

Dep. Variable:	ApprovalStatus_1	No. Observations:	482
Model:	Logit	Df Residuals:	454
Method:	MLE	Df Model:	27
Date:	Fri, 23 Aug 2024	Pseudo R-squ.:	0.5731
Time:	17:38:59	Log-Likelihood:	-141.22
converged:	True	LL-Null:	-330.84
Covariance Type:	nonrobust	LLR p-value:	8.318e-64

	coef	std err	z	P>|z|	[0.025	0.975]
const	-2.5168	1.214	-2.073	0.038	-4.897	-0.137
Age	-0.0050	0.016	-0.322	0.748	-0.036	0.026
Debt	-0.0009	0.036	-0.025	0.980	-0.072	0.071
Log_Income	0.1231	0.057	2.174	0.030	0.012	0.234
Log_YearsEmployed	0.1016	0.130	0.781	0.435	-0.153	0.357
Log_CreditScore	0.6209	0.293	2.120	0.034	0.047	1.195
Gender_Male	-0.3208	0.367	-0.874	0.382	-1.040	0.399
BankCustomer_NoBankAccount	-0.9768	0.393	-2.485	0.013	-1.747	-0.207
Industry_ConsumerDiscretionary	-1.1678	0.919	-1.270	0.204	-2.970	0.634
Industry_ConsumerStaples	-0.9632	0.868	-1.110	0.267	-2.664	0.737
Industry_Education	-0.8535	1.188	-0.719	0.472	-3.181	1.474
Industry_Energy	-0.6697	0.774	-0.865	0.387	-2.187	0.848

Industry_Financials	0.5938	0.840	0.707	0.480	-1.053	2.241
Industry_Industrials	2.0767	0.840	2.472	0.013	0.430	3.723
Industry_InformationTechnology	3.0669	1.003	3.057	0.002	1.101	5.033
Industry_Materials	1.4715	0.780	1.886	0.059	-0.058	3.001
Industry_RealEstate	0.2920	0.962	0.303	0.762	-1.594	2.178
Industry_Research	-2.1274	1.538	-1.383	0.167	-5.142	0.888
Industry_Transport	4.2646	1.223	3.486	0.000	1.867	6.663
Industry_Utilities	3.5452	1.162	3.050	0.002	1.267	5.823
Ethnicity_black	0.8830	0.701	1.260	0.208	-0.491	2.257
Ethnicity_latino	0.9980	1.098	0.909	0.363	-1.153	3.149
Ethnicity_others	3.6545	1.233	2.965	0.003	1.239	6.070
Ethnicity_white	0.6401	0.654	0.980	0.327	-0.641	1.921
PriorDefault_1	3.6923	0.422	8.758	0.000	2.866	4.519
Employed_1	0.1639	0.491	0.334	0.738	-0.798	1.126
DriversLicense_1	-0.0156	0.335	-0.047	0.963	-0.671	0.640
Citizen_other means	0.4546	0.523	0.869	0.385	-0.571	1.480

```python
# Set significance level
significance_level = 0.05

# Identify insignificant features
insignificant_features = model.pvalues[model.pvalues > significance_level].index

# Drop insignificant features (excluding the intercept)
insignificant_features = insignificant_features.drop('const', errors='ignore')
X_train_sig = X_nomulti_train_cns.drop(columns=insignificant_features)

# Fit the reduced model
model_sig = sm.Logit(Y_train,X_train_sig ).fit()

# Print summary of the reduced model
print(model_sig.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.307880
         Iterations 7
```

Logit Regression Results

Dep. Variable:	ApprovalStatus_1	No. Observations:	482
Model:	Logit	Df Residuals:	473
Method:	MLE	Df Model:	8
Date:	Wed, 14 Aug 2024	Pseudo R-squ.:	0.5514
Time:	16:02:10	Log-Likelihood:	-148.40
converged:	True	LL-Null:	-330.84
Covariance Type:	nonrobust	LLR p-value:	6.031e-74

	coef	std err	z	P>|z|	[0.025	0.975]
const	-3.8854	0.378	-10.283	0.000	-4.626	-3.145
Log_Income	0.1479	0.052	2.845	0.004	0.046	0.250
Log_CreditScore	0.8901	0.223	3.998	0.000	0.454	1.326
Industry_Industrials	1.2659	0.570	2.221	0.026	0.149	2.383
Industry_InformationTechnology	2.2121	0.769	2.878	0.004	0.706	3.719
Industry_Transport	3.4799	0.988	3.522	0.000	1.544	5.416
Industry_Utilities	2.6458	0.828	3.196	0.001	1.023	4.268
Ethnicity_others	1.8914	0.796	2.376	0.018	0.331	3.452
PriorDefault_1	3.8739	0.370	10.458	0.000	3.148	4.600

```
# Splits to be tested
splits = [0.2, 0.25, 0.3, 0.4]

# Loop through each dataset
for name, data in datasets.items():
    X = data.drop(['ApprovalStatus_1'], axis=1)  # Assuming 'ApprovalStatus_1' is your label column
    y = data['ApprovalStatus_1']

    print(f"Results for dataset: {name}")

    # Loop through each split
    for split in splits:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=split, random_state=42)

        # Feature scaling
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        # Logistic Regression model
        logistic = LogisticRegression(C=1e9, random_state=42)  # Using a large C value to avoid regularization
        logistic.fit(X_train_scaled, y_train)
```

```
        # Predictions and metrics
        y_pred = logistic.predict(X_test_scaled)
        accuracy = accuracy_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred)

        print(f"Train-Test Split: {int((1-split)*100)}-{int(split*100)} | Accuracy: {accuracy:.3f} | F1 Score: {f1:.3f}")

    print("\n" + "-"*50 + "\n")
```

```
Results for dataset: data_dummies_clean
Train-Test Split: 80-20 | Accuracy: 0.841 | F1 Score: 0.841
Train-Test Split: 75-25 | Accuracy: 0.850 | F1 Score: 0.840
Train-Test Split: 70-30 | Accuracy: 0.826 | F1 Score: 0.820
Train-Test Split: 60-40 | Accuracy: 0.826 | F1 Score: 0.811

--------------------------------------------------

Results for dataset: data_dummies_nomulti
Train-Test Split: 80-20 | Accuracy: 0.841 | F1 Score: 0.841
Train-Test Split: 75-25 | Accuracy: 0.844 | F1 Score: 0.832
Train-Test Split: 70-30 | Accuracy: 0.841 | F1 Score: 0.834
Train-Test Split: 60-40 | Accuracy: 0.826 | F1 Score: 0.811

--------------------------------------------------

Results for dataset: data_dummies_sig
Train-Test Split: 80-20 | Accuracy: 0.848 | F1 Score: 0.853
Train-Test Split: 75-25 | Accuracy: 0.861 | F1 Score: 0.859
Train-Test Split: 70-30 | Accuracy: 0.855 | F1 Score: 0.853
Train-Test Split: 60-40 | Accuracy: 0.859 | F1 Score: 0.854

--------------------------------------------------
```

Logistic		Accuracy		
	80-20	75-25	70-30	60-40
data_dummies_clean	0.841	0.85	0.826	0.826
data_dummies_nomulti	0.841	0.844	0.841	0.826
data_dummies_sig	0.848	0.861	0.855	0.859

KNN_Classifier

```
[ ] from sklearn.model_selection import train_test_split
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.metrics import accuracy_score, f1_score
    from sklearn.preprocessing import StandardScaler

    # Your datasets and their corresponding names
    datasets = {
        'data_dummies_clean': data_dummies_clean,
        'data_dummies_nomulti': data_dummies_nomulti,
        'data_dummies_sig': data_dummies_sig
    }

    # Splits to be tested
    splits = [0.2, 0.25, 0.3, 0.4]

    # Loop through each dataset
    for name, data in datasets.items():
        X = data.drop(['ApprovalStatus_1'], axis=1)  # Assuming 'ApprovalStatus_1' is your label column
        y = data['ApprovalStatus_1']

        print(f"Results for dataset: {name}")

        # Loop through each split
        for split in splits:
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=split, random_state=42)
```

```python
# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# KNN model
knn = KNeighborsClassifier(n_neighbors=5)  # You can adjust n_neighbors as needed
knn.fit(X_train_scaled, y_train)

# Predictions and metrics
y_pred = knn.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Train-Test Split: {int((1-split)*100)}-{int(split*100)} | Accuracy: {accuracy:.3f} | F1 Score

t("\n" + "-"*50 + "\n")
```

```
for dataset: data_dummies_clean
st Split: 80-20 | Accuracy: 0.768 | F1 Score: 0.750
st Split: 75-25 | Accuracy: 0.780 | F1 Score: 0.743
st Split: 70-30 | Accuracy: 0.802 | F1 Score: 0.771
st Split: 60-40 | Accuracy: 0.801 | F1 Score: 0.758

------------------------------------------

for dataset: data_dummies_nomulti
st Split: 80-20 | Accuracy: 0.783 | F1 Score: 0.766
st Split: 75-25 | Accuracy: 0.798 | F1 Score: 0.765
st Split: 70-30 | Accuracy: 0.807 | F1 Score: 0.778
st Split: 60-40 | Accuracy: 0.808 | F1 Score: 0.767

------------------------------------------
```

```
Results for dataset: data_dummies_clean
Train-Test Split: 80-20 | Accuracy: 0.768 | F1 Score: 0.750
Train-Test Split: 75-25 | Accuracy: 0.780 | F1 Score: 0.743
Train-Test Split: 70-30 | Accuracy: 0.802 | F1 Score: 0.771
Train-Test Split: 60-40 | Accuracy: 0.801 | F1 Score: 0.758

--------------------------------------------------

Results for dataset: data_dummies_nomulti
Train-Test Split: 80-20 | Accuracy: 0.783 | F1 Score: 0.766
Train-Test Split: 75-25 | Accuracy: 0.798 | F1 Score: 0.765
Train-Test Split: 70-30 | Accuracy: 0.807 | F1 Score: 0.778
Train-Test Split: 60-40 | Accuracy: 0.808 | F1 Score: 0.767

--------------------------------------------------

Results for dataset: data_dummies_sig
Train-Test Split: 80-20 | Accuracy: 0.812 | F1 Score: 0.819
Train-Test Split: 75-25 | Accuracy: 0.821 | F1 Score: 0.795
Train-Test Split: 70-30 | Accuracy: 0.831 | F1 Score: 0.811
Train-Test Split: 60-40 | Accuracy: 0.844 | F1 Score: 0.840

--------------------------------------------------
```

DecisionTree_classifier

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, f1_score
from sklearn.preprocessing import StandardScaler

# Your datasets and their corresponding names
datasets = {
    'data_dummies_clean': data_dummies_clean,
    'data_dummies_nomulti': data_dummies_nomulti,
    'data_dummies_sig': data_dummies_sig
}

# Splits to be tested
splits = [0.2, 0.25, 0.3, 0.4]
```

```
hrough each split
t in splits:
ain, X_test, y_train, y_test = train_test_split(X, y, test_size=split, random_state=42)

ature scaling
er = StandardScaler()
ain_scaled = scaler.fit_transform(X_train)
st_scaled = scaler.transform(X_test)

cision Tree model
 DecisionTreeClassifier(random_state=42)  # You can adjust hyperparameters as needed
it(X_train_scaled, y_train)

edictions and metrics
ed = dt.predict(X_test_scaled)
racy = accuracy_score(y_test, y_pred)
 f1_score(y_test, y_pred)

t(f"Train-Test Split: {int((1-split)*100)}-{int(split*100)} | Accuracy: {accuracy:.3f} | F1 Score:

n" + "-"*50 + "\n")
```

```
Results for dataset: data_dummies_clean
Train-Test Split: 80-20 | Accuracy: 0.783 | F1 Score: 0.766
Train-Test Split: 75-25 | Accuracy: 0.792 | F1 Score: 0.769
Train-Test Split: 70-30 | Accuracy: 0.812 | F1 Score: 0.791
Train-Test Split: 60-40 | Accuracy: 0.797 | F1 Score: 0.774

------------------------------------------------

Results for dataset: data_dummies_nomulti
Train-Test Split: 80-20 | Accuracy: 0.768 | F1 Score: 0.754
Train-Test Split: 75-25 | Accuracy: 0.775 | F1 Score: 0.748
Train-Test Split: 70-30 | Accuracy: 0.812 | F1 Score: 0.789
Train-Test Split: 60-40 | Accuracy: 0.801 | F1 Score: 0.777

------------------------------------------------

Results for dataset: data_dummies_sig
Train-Test Split: 80-20 | Accuracy: 0.804 | F1 Score: 0.803
Train-Test Split: 75-25 | Accuracy: 0.827 | F1 Score: 0.815
Train-Test Split: 70-30 | Accuracy: 0.841 | F1 Score: 0.829
Train-Test Split: 60-40 | Accuracy: 0.859 | F1 Score: 0.848
```

Decision Tree	80-20	75-25	70-30	60-40
Data_Dummies_Clean	0.783	0.792	0.812	0.797
Data_Dummies_No Multi	0.768	0.775	0.812	0.801
Data_Dummies_Sig	0.804	0.827	0.841	0.859

RandomForest

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score
from sklearn.preprocessing import StandardScaler

# Your datasets and their corresponding names
datasets = {
    'data_dummies_clean': data_dummies_clean,
    'data_dummies_nomulti': data_dummies_nomulti,
    'data_dummies_sig': data_dummies_sig
}

# Splits to be tested
splits = [0.2, 0.25, 0.3, 0.4]

# Loop through each dataset
for name, data in datasets.items():
    X = data.drop(['ApprovalStatus_1'], axis=1)  # Assuming 'ApprovalStatus_1' is your label column
```

```
# Loop through each dataset
for name, data in datasets.items():
    X = data.drop(['ApprovalStatus_1'], axis=1)  # Assuming 'ApprovalStatus_1' is your label column
    y = data['ApprovalStatus_1']

    print(f"Results for dataset: {name}")

    # Loop through each split
    for split in splits:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=split, random_state=42)
```

```
        # Feature scaling
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        # Random Forest model
        rf = RandomForestClassifier(random_state=42)  # You can adjust hyperparameters as needed
        rf.fit(X_train_scaled, y_train)

        # Predictions and metrics
        y_pred = rf.predict(X_test_scaled)
        accuracy = accuracy_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred)

        print(f"Train-Test Split: {int((1-split)*100)}-{int(split*100)} | Accuracy: {accuracy:.3f} | F1 Score: {f1:.3f}")

    print("\n" + "-"*50 + "\n")
```

```
Results for dataset: data_dummies_clean
Train-Test Split: 80-20 | Accuracy: 0.841 | F1 Score: 0.838
Train-Test Split: 75-25 | Accuracy: 0.838 | F1 Score: 0.825
Train-Test Split: 70-30 | Accuracy: 0.860 | F1 Score: 0.851
Train-Test Split: 60-40 | Accuracy: 0.862 | F1 Score: 0.849

--------------------------------------------------

Results for dataset: data_dummies_nomulti
Train-Test Split: 80-20 | Accuracy: 0.841 | F1 Score: 0.838
Train-Test Split: 75-25 | Accuracy: 0.844 | F1 Score: 0.830
Train-Test Split: 70-30 | Accuracy: 0.860 | F1 Score: 0.851
Train-Test Split: 60-40 | Accuracy: 0.862 | F1 Score: 0.852

--------------------------------------------------

Results for dataset: data_dummies_sig
Train-Test Split: 80-20 | Accuracy: 0.819 | F1 Score: 0.823
Train-Test Split: 75-25 | Accuracy: 0.844 | F1 Score: 0.838
Train-Test Split: 70-30 | Accuracy: 0.841 | F1 Score: 0.837
Train-Test Split: 60-40 | Accuracy: 0.862 | F1 Score: 0.855

--------------------------------------------------
```

Random Forest	80-20	75-25	70-30	60-40
Data_Dummies_Clean	0.841	0.838	0.86	0.862
Data_Dummies_No Multi	0.841	0.844	0.86	0.862
Data_Dummies_Sig	0.819	0.844	0.841	0.862

Bagging

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import accuracy_score, f1_score
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier  # Using Decision Tree as the base estimator

# Your datasets and their corresponding names
datasets = {
    'data_dummies_clean': data_dummies_clean,
    'data_dummies_nomulti': data_dummies_nomulti,
    'data_dummies_sig': data_dummies_sig
}

# Splits to be tested
splits = [0.2, 0.25, 0.3, 0.4]
```

```
# Loop through each dataset
for name, data in datasets.items():
    X = data.drop(['ApprovalStatus_1'], axis=1)  # Assuming 'ApprovalStatus_1' is your label column
    y = data['ApprovalStatus_1']

    print(f"Results for dataset: {name}")

    # Loop through each split
    for split in splits:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=split, random_state=42)

        # Feature scaling
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)
```

```
Bagging model (using Decision Tree as the base estimator)
gging = BaggingClassifier(estimator=DecisionTreeClassifier(random_state=42), random_state=42)
gging.fit(X_train_scaled, y_train)

Predictions and metrics
pred = bagging.predict(X_test_scaled)
curacy = accuracy_score(y_test, y_pred)
 = f1_score(y_test, y_pred)

int(f"Train-Test Split: {int((1-split)*100)}-{int(split*100)} | Accuracy: {accuracy:.3f} | F1 Score: {
```

```
Results for dataset: data_dummies_clean
Train-Test Split: 80-20 | Accuracy: 0.783 | F1 Score: 0.762
Train-Test Split: 75-25 | Accuracy: 0.850 | F1 Score: 0.833
Train-Test Split: 70-30 | Accuracy: 0.850 | F1 Score: 0.834
Train-Test Split: 60-40 | Accuracy: 0.841 | F1 Score: 0.820

--------------------------------------------------

Results for dataset: data_dummies_nomulti
Train-Test Split: 80-20 | Accuracy: 0.797 | F1 Score: 0.774
Train-Test Split: 75-25 | Accuracy: 0.855 | F1 Score: 0.837
Train-Test Split: 70-30 | Accuracy: 0.865 | F1 Score: 0.851
Train-Test Split: 60-40 | Accuracy: 0.822 | F1 Score: 0.798

--------------------------------------------------

Results for dataset: data_dummies_sig
Train-Test Split: 80-20 | Accuracy: 0.826 | F1 Score: 0.826
Train-Test Split: 75-25 | Accuracy: 0.838 | F1 Score: 0.831
Train-Test Split: 70-30 | Accuracy: 0.845 | F1 Score: 0.842
Train-Test Split: 60-40 | Accuracy: 0.862 | F1 Score: 0.855

--------------------------------------------------
```

Bagging	80-20	75-25	70-30	60-40
Data_Dummies_Clean	0.783	0.85	0.85	0.841
Data_Dummies_No Multi	0.797	0.855	0.865	0.822
Data_Dummies_Sig	0.826	0.838	0.845	0.862

Boosting

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, f1_score
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier  # Using Decision Tree as the base estimator

# Your datasets and their corresponding names
datasets = {
    'data_dummies_clean': data_dummies_clean,
    'data_dummies_nomulti': data_dummies_nomulti,
    'data_dummies_sig': data_dummies_sig
}

# Splits to be tested
splits = [0.2, 0.25, 0.3, 0.4]
```

```
# Loop through each dataset
for name, data in datasets.items():
    X = data.drop(['ApprovalStatus_1'], axis=1)  # Assuming 'ApprovalStatus_1' is your label column
    y = data['ApprovalStatus_1']

    print(f"Results for dataset: {name}")

    # Loop through each split
    for split in splits:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=split, random_state=42)

        # Feature scaling
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)
```

```
    X_test_scaled = scaler.transform(X_test)

    # AdaBoost model (using Decision Tree as the base estimator)
    boosting = AdaBoostClassifier(estimator=DecisionTreeClassifier(random_state=42), random_state=42)
    boosting.fit(X_train_scaled, y_train)

    # Predictions and metrics
    y_pred = boosting.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    print(f"Train-Test Split: {int((1-split)*100)}-{int(split*100)} | Accuracy: {accuracy:.3f} | F1 Score: {f1:.3f}")
```

```
Results for dataset: data_dummies_clean
Train-Test Split: 80-20 | Accuracy: 0.775 | F1 Score: 0.760
Train-Test Split: 75-25 | Accuracy: 0.803 | F1 Score: 0.782
Train-Test Split: 70-30 | Accuracy: 0.816 | F1 Score: 0.796
Train-Test Split: 60-40 | Accuracy: 0.804 | F1 Score: 0.782

--------------------------------------------------

Results for dataset: data_dummies_nomulti
Train-Test Split: 80-20 | Accuracy: 0.761 | F1 Score: 0.744
Train-Test Split: 75-25 | Accuracy: 0.786 | F1 Score: 0.764
Train-Test Split: 70-30 | Accuracy: 0.812 | F1 Score: 0.796
Train-Test Split: 60-40 | Accuracy: 0.801 | F1 Score: 0.777

--------------------------------------------------

Results for dataset: data_dummies_sig
Train-Test Split: 80-20 | Accuracy: 0.797 | F1 Score: 0.794
Train-Test Split: 75-25 | Accuracy: 0.809 | F1 Score: 0.800
Train-Test Split: 70-30 | Accuracy: 0.836 | F1 Score: 0.825
Train-Test Split: 60-40 | Accuracy: 0.830 | F1 Score: 0.820
```

Bosting	80-20	75-25	70-30	60-40
Data_Dummies_Clean	0.775	0.803	0.816	0.804
Data_Dummies_No Multi	0.761	0.786	0.812	0.801
Data_Dummies_Sig	0.797	0.809	0.836	0.83

Conclusion

	All Data				Data with NoMulticolinearity Features				Data with Significant Features			
	80-20	75-25	70-30	60-40	80-20	75-25	70-30	60-40	80-20	75-25	70-30	60-40
Logistic	0.841	0.844	0.841	0.826	0.841	0.844	0.841	0.826	0.848	0.861	0.855	0.859
Knn	0.783	0.798	0.807	0.808	0.783	0.798	0.807	0.808	0.812	0.821	0.831	0.859
SVM	0.826	0.832	0.831	0.855	0.826	0.832	0.831	0.855	0.841	0.855	0.855	0.859
DecisionTree	0.768	0.775	0.812	0.801	0.768	0.775	0.812	0.801	0.804	0.827	0.841	0.859
RandomForest	0.841	0.844	0.86	0.862	0.841	0.844	0.86	0.862	0.819	0.844	0.841	0.859
Bagging	0.797	0.855	0.865	0.822	0.797	0.855	0.865	0.822	0.826	0.838	0.845	0.859
Boosting	0.761	0.786	0.812	0.801	0.761	0.786	0.812	0.801	0.797	0.809	0.836	0.859